一种高效的并行挖掘频繁序列的算法*)

佘春东 吴 跃 孙世新 李 磊 车著明
(电子科技大学计算机科学与工程学院 成都610054)

摘 要 序列模式发现在数据挖掘领域中的地位越来越重要,本文首先介绍了频繁序列挖掘模式的基本概念,然后基于投影树算法,给出了其数据并行模式和任务并行模式,接着进行了算法的复杂性分析,我们的实验证明这些算法都能获得较好的加速比,而且任务并行模式具有更好的可扩展性。

关键词 并行处理,任务分解,投影算法,频繁序列模式

An Efficient Parallel Algorithm for Mining Frequent Sequences

SHE Chun-Dong WU Yue SUN Shi-Xin LI Lei CHE Zhu-Ming
(School of computer Science and Engineering, UESTC, Chengdu 610054)

Abstract Discovery of sequential patterns is becoming increasingly essential in data mining field. The paper briefly introduces the basic concept of frequent sequence mining and presents the data parallel formulation and task parallel formulation of tree-projection based algorithm. Moreover, the complexity analysis of the algorithm is given. Our experiment shows that these two algorithms are capable of achieving good speedups. However, the task parallel formulation is more scalable than the data parallel one.

Keywords Parallel processing, Task decomposition, Projection algorithm, Frequent sequential patterns

1 引言

最近,数据挖掘技术发展迅速,基因分析中许多重要的知识发现任务需要对DNA和蛋白质序列进行分析,从而发现频繁出现的核酸和氨基酸子序列。这类任务中最耗时的操作是计算序列数据库中所有子序列(称为序列模式)的发生频度。然而,由于这类序列模式呈指数级增长,于是人们通过加上各种临时约束和仅考虑满足用户指定的最小支持度的候选集以减少问题的复杂度[1,3,5]。即使如此,找出所有的序列模式仍需要大量的计算资源,使得采用并行处理技术是解决此类问题的理想手段。

发现序列模式的算法主要有三类,第一类算法[1]是基于Apriori[3]算法的,这类算法利用候选模式产生机制来发现频繁的序列模式;第二类算法[6,7]则是将频繁序列分解成若干等价类,通过使用垂直的数据库方式和集合的交运算来计算序列的频度;第三类算法使用投影树技术以及将原始数据库划分为特定模式的子数据库的方式来发现序列模式。总的来说,基于投影的频繁模式发现算法在性能上明显优于其它算法,但仍然需要大量的计算时间。

目前人们开发了许多高效的基于候选集产生和计数框架的发现频繁集和序列模式的算法,以及该类可扩展的针对共享存储和分布式存储的并行算法。但人们对基于等价类和投影算法的并行化问题的关注相对较少,而且该类算法现有的并行模式只是针对共享存储体系结构的。然而,这些算法产生的任务图具有的不规则和非结构化特征以及它们所运行的子数据库的相互重叠的特性,使得开发适于分布式存储并行计算机的高效、可扩展的并行算法具有较大的挑战性。

2 串行序列挖掘

2.1 序列挖掘的基本概念

令 $I=\{i_1,i_2,\cdots,i_n\}$ 是所有项目的集合。一个项目集就是 I 的一个子集;而一个序列 $s=\langle t_1,t_2,\cdots,t_l\rangle$ 则定义为一个排了序的项目列表,其中 $1\leqslant j\leqslant l$, $t_j\subseteq I$;而一个序列数据库就是一个包含很多序列的集合。序列 s 的长度定义为 s 中的项目数,用 $|s|$ 表示。对于给定的序列数据库 D,$|D|$ 表示 D 中序列的个数。假设 I 中的项目可以按字典序排列,我们可以用从1开始的连续整数来表示排了序的项目。

如果存在 l 个整数 $i_1,i_2,\cdots,i_l$,满足 $1\leqslant i_1<i_2,\cdots<i_l$,且 $t_j\subseteq ti_j(j=1,2,\cdots,l)$,则称序列 $s=\langle t_1,t_2,\cdots,t_l\rangle$ 为序列 $\underline{s}=\langle \underline{t_1},\underline{t_2},\cdots,\underline{t_m}\rangle$ $(l\leqslant m)$ 的子序列,记为 $s\subseteq\underline{s}$,并认为 s 支持 $\underline{s}$,给定一个序列 s,令 $Ds\subseteq D$ 为 D 中支持 s 的序列集合,则 s 的支持度定义为 $|Ds|/|D|$,表示为 $\sigma D(s)$,易知,对于任意 $s\in D$,有 $0\leqslant\sigma D(s)\leqslant 1$。

对于给定最小支持度约束的发现频繁序列模式问题,文[1]中定义如下:

定义1(序列模式挖掘) 给定序列数据库 D 和用户指定的参数 $\sigma(0\leqslant\sigma\leqslant 1)$,找出 D 中所有至少被 $\sigma|D|$ 个序列支持的序列。用户指定的参数 σ 称为最小支持度约束,而找出的子序列称为频繁序列模式。

2.2 串行投影树算法

*)本文得到中国科学院知识创新工程方向性研究项目基金(名称:大型数字对象应用环境及其并行模拟,批准号:KGCX2-JG-09)和总装备部试验技术项目基金的资助。**佘春东** 博士研究生,高级工程师,主要研究方向为并行计算与数据库技术。**吴 跃** 教授,博士生导师,主要研究方向为数据库技术。**孙世新** 教授,博士生导师,主要研究方向为并行计算技术。**李 磊** 研究员,博士生导师,主要研究方向为数据挖掘与数据库技术。**车著明** 高级工程师,主要研究方向为数据处理技术。

本文的并行算法是基于文[2]提出的用于频繁项目集挖掘的投影树算法。该算法利用一个按字典序排列的投影树数据结构来表示所有项目集的集合。树中的某个节点对应于一个特定的项目集，而该节点在树中所处的层数对应于项目集的长度。

只有当对应于频繁项目集的节点生成时，才可用投影树算法以广度优先或深度优先的方式对生成树进行扩展。投影树算法的一个主要特征是通过使用数据库投影方法，在第 k－1层的双亲节点对原始事务数据进行投影，来决定在算法的第 k 次迭代过程中所产生的候选项目集的支持度。具体做法是对于树中的每一个节点，保留一张项目表，表中的每个项目可以在对应于后辈节点的项目集中找到。这些项目称为节点的动态项目，当一个事务投影到某个节点上时，只有出现在动态项目表中的那些项目才保留下来，而剩余的项目则被剔除。

2.3 基于投影树算法的序列模式挖掘

发现序列模式的投影树算法也是使用一个字典序的树结构来表示所有可能的序列。树中的每个节点对应于一个特殊的序列而且第 k 层的节点对应于长度为 k 的序列。序列节点与其子节点之间的关系本质上类似于原投影树算法中的项目集之间的关系。然而，有所不同的是，每个节点有两种类型的子节点，其一是通过在该序列的最后一个项目集中增加一个项目来获得，但所增加项目的字典序必须比最后一个项目集中的所有项目要大；另一种类型的子节点可以通过在序列的最后增加一个大小为1的新项目来获得，此时对项目的字典序没有限制。这两种类型节点的获得过程分别称为项目集扩展和序列扩展。

类似地，只有当对应于频繁序列的节点生成时才对字典树进行扩展。此外，可以通过使用双层候选序列生成方式来提高算法的效率，在此方式中，第 k－1层的节点可以用于生成第 k＋1层的候选序列。通过将数据库序列不断地沿着从根节点到第 k－1层的节点的路径进行投影操作，来确定候选序列的频度。由于序列节点可以通过项目集和序列扩展两种方式来进行扩展，因而必须保持两个动态项目集，其一用于项目集扩展，另一用于序列扩展。在投影过程中不是这两个集合中的项目则被剔出。

不同长度为 k＋1的序列的频度的确定如下所述。每个节点，表示为序列 $s=\langle t_1,t_2,\cdots,t_m\rangle$，在第 k＋1层保留四个计数矩阵。其一用于计算候选序列扩展 $\langle t_1,t_2,\cdots,(t_m,i,j)\rangle$ 的频度，其中项目 i 和 j 是节点的动态项目集扩展；其二用于计算候选序列扩展 $\langle t_1,t_2,\cdots,t_m,(i),(j)\rangle$ 的频度，其中项目 i 和 j 是节点的动态序列扩展；其三用于计算候选序列扩展 $\langle t_1,t_2,\cdots,t_m,(i,j)\rangle$ 的频度，其中项目 i 和 j 是节点的动态序列扩展，并且项目 j 的字典序大于 i。最后一个矩阵用于计算候选序列扩展 $\langle t_1,t_2,\cdots,(t_m,i),(j)\rangle$ 的频度，其中项目 i 是节点的动态项目集扩展，项目 j 是节点的动态序列扩展。

一旦每个序列都投影到第 k－1层上某个特定的节点，算法则立即更新相应矩阵的频度。可以通过使用稀疏矩阵使得更新效率得以提高，该矩阵的每行对应于某个项目集，而每列则对应于出现在序列中的不同项目。

3 并行序列挖掘

算法的目标机是分布式存储的并行计算机，我们假定消息传递是处理器间通信的唯一手段，此外，还假定数据库太大无法直接装入主存，这样的假定使得面临大规模数据集时算法具有较好的可扩展性。对于此类问题一般有两种并行化的方法，一种是数据并行模式，另一种是任务并行模式。

3.1 基于投影树算法的序列挖掘的数据并行模式

数据并行模式的思想是将确定树中不同序列节点频度所需的计算量在不同的处理器间进行划分，本质上类似于文[3]中提出的数据分发方法。下面给出并行算法的工作机制。令 p 是处理器的总数，则将原始数据库初始划分为 p 块大小相等的部分，每块分配给某个不同的处理器，并存储在其本地磁盘上。这些处理器协同工作，以广度优先扩展的方式每次将投影树扩展一层。而计算相应于树中第 k＋1层节点的候选序列的频度分两步实施，第一步，每个处理器只是将数据库的本地子集投影到第 k－1层的节点从而计算出序列的局部频度，算法的第二步通过全局规约操作将各自的频度累加起来得到全局频度。各处理器利用这些全局频度值来确定第 k＋1层的节点哪些满足最小支持度约束。应该注意的是，所有处理器使用的数据结构是相同的，与串行算法中所建立的树一样。

一般来说，由于数据库在各处理器间进行了等分，总的计算量会得到均衡分配。然而，如果某些处理器分配的子数据库所包含的序列模式相对较多，则会导致负载的不平衡。

3.2 基于投影树算法的序列挖掘的任务并行模式

投影树算法通过将数据库序列投影到第 k－1层的不同节点来计算第 k＋1层的候选模式的频度。一旦数据库已经被投影到这些节点上，则实际的频度计算可以在每个节点独立地进行，于是第 k－1层各节点的计算量就成了独立的任务，通过将这些任务分配给不同的处理机从而进行算法的并行化。

可以使用水平和垂直模式来分配与树中不同节点相关联的任务，在水平模式中，每层的各节点依次生成，然后该层的节点任务被分配给不同的处理机，在不同层次的分解是不同的，而在垂直模式中，每个处理器分配到对应于 k＋1层某些特定的节点的一系列任务，然后独立地生成扎根于这些节点的所有子树。在总体上，垂直模式在分布存储体系结构的并行机上的运行性能要优于垂直模式，而水平模式则更适于共享存储体系，下面我们就提出基于垂直模式的任务并行分解方式，该模式以下述方式在处理器间分配任务。首先，利用3.1节所描述的数据并行算法对投影树进行扩张，当扩展到第 k＋1层后，第 k 层的不同节点被分配给各处理器，然后每个处理器就以分配给它的不同节点为树根来生成子森林。注意算法是对第 k 层节点进行分配(而非第 k＋1层)，因为对第 k＋2层的候选集进行频度计数时，需要将数据库投影到第 k 层节点上。

为了使每个处理器接下来能独立地进行工作，它必须访问数据库中可能会支持分配给该处理器节点所对应模式的序列。每个处理器 P_i 所要访问的数据库序列按下述方法确定。令 S_{iw} 为分配给 P_i 的节点的集合，A_i 为 S_{iw} 中所有动态项目节点的联合，B_{iw} 为 S_{iw} 中所有频繁序列节点所包含项目的集合。于是，每个处理器要访问所有满足以下条件的序列，该序列所包含的项目集是 $C_i=A_i\cup B_i$ 的子集，此外，由于它只计算对应于 S_{iw} 后辈节点所对应的频繁模式，因此只需保留原序列中出现在 C_i 中的项目。

在此算法中，一旦确定了第 k 层节点的分配方案，也就确定了集合 $C_0,C_1,\cdots C_{p-1}$ 并以广播方式发送给各处理器，然后每个处理器开始读取数据库的本地部分，并将它切成 p 块，每个处理器一块，并将它发送到相应的处理器。当处理器收到与本地树中相匹配的序列时，则将它们写入盘中并独立地进

行节点扩展。

4 算法的性能分析

一个并行算法的效率可以通过使用一系列诸如并行运行时间、加速比以及最大并发度等尺度来衡量[4]，一个具有较好的可扩展性的并行算法随着处理器数目的增加和问题规模的增加仍将保持其较高的并行效率。由于基于投影树的串行算法本身已很复杂而且其运行时间取决于最小支持度和输入数据库的各种特性(比如，不同项目的个数、所包含的频繁序列的个数等)，因此，文中在对并行算法的性能进行分析时作了一些必要的简化。

4.1 数据并行模式

由于投影树的深度取决于最小支持度以及数据集的特性，我们的分析集中于在计算对应于投影树中第k+2层的候选序列的频度时算法的并行效率。我们知道第k+2层的候选序列的频度的计算是通过将本地存储的数据库序列投影到第k层的节点上，并使用四个计算矩阵来计算其频度，然后通过全局规约操作将这些本地频度累加起来。这些矩阵的大小取决于树中每个节点动态项目的个数，而且在最坏情况下其上限为$|I|$(I表示数据库中不同项目的集合)，因此，令m_k是树中第k层节点的个数，则规约操作中涉及的数据量为$O(|I|^2m_k)$，远大于处理器的数量，于是每个处理器在规约操作过程中的通信开销为$T1=O(|I|^2m_k)$。

串行算法用于计算第k+2层候选序列频度所需的时间T2取决于数据库中的序列数、不同项目的个数以及第k层节点的个数m_k，即$T2=f(|D|,|I|,m_k)$。注意到m_k取决于$|I|$以及最小支持度σ，随着$|I|$的增加或σ的减少，序列模式的数量也将增加，于是m_k也将增加。由于数据库中的每个序列都有可能被投影到第k层的节点上，因此在最坏的情况下，$f(|D|,|I|,m_k)=O(|D||I|^2m_k)$。而由于将数据库投影到某个节点上所需的工作会被其他有同样先辈的节点分担，于是$T2$可表示为$O(|D|^\alpha|I|^2m_k)$，其中$0<\alpha<1.0$。虽然难以估算α的值，但将某个特定的序列投影到树中第k层的所有节点上所需的总工作量对于串行和并行算法是相同的。

令处理机的个数为p，由以上的讨论可知，将数据库投影到第k层上的节点以及确定第$k+2$层节点的频度所需的时间理论上为$Tp=T2/p+T1=O(|D|^\alpha|I|^2m_k)/p+O(|I|^2m_k)$。

4.2 任务并行模式

基于投影树算法的任务并行模式的性能主要取决于进行任务分解以及在处理器间重新划分数据库所需的时间。一旦任务分配方案确定后，算法才进行数据库的再划分，再划分的通信开销取决于每个处理器所要接收的数据库的规模。在理想状态下，由于每个处理器平均收到的数据量大小为数据库的$1/p$，因此通信时间为$O(|D|/p)$，然而，由于总会有部分的数据重叠，实际的通信开销可能会高一些。为了便于分析，我们假定每个处理器需要接收的数据库序列的平均数为$\beta|D|/p(\beta\geqslant1)$，于是任务分解所需的时间为$T=O(\beta|D|/p)$。而每个处理器$P_i$生成子森林所需的时间$Ti$取决于本地数据库的大小$\beta|D|/p$、其后辈节点的总数$d_i$以及$|I|$($I$表示数据库中不同项目的集合)，根据4.1节的类似推理可知：

$$Ti=f(\beta|D|/p,|I|,d_i)=O((\beta|D|/p)^\alpha|I|^2d_i)$$

若$d_i|I|\gg1$，则算法用于任务分解的时间远小于处理器生成本地子森林的时间，这时任务并行模式将是非常有效的。

5 实验结果分析

实验是在曙光TC1700并行机上进行的，该机型配有4个节点，每个节点配有2个处理器、512MB的内存和18GB的硬盘；OS为Linux，并行计算环境为MPI。序列数据库的规模为100k，最小支持度为1%，在千兆以太网环境下的性能数据如表1所示。

表1

	数据并行模式		任务并行模式	
处理器个数	执行时间	加速比	执行时间	加速比
2	33.50	1.00	31.70	1.00
4	18.82	1.78	17.04	1.86
6	13.67	2.45	11.40	2.78
8	10.91	3.07	8.85	3.58

我们定义处理器个数为2时算法的加速比为1，由表1可以看出，随着处理器数量的增加，两种并行算法的执行时间都明显减少，并取得了较好的加速比。同时我们注意到随之处理器数目的增加，数据并行模式的并行效率随着下降，这是因为分配给每个处理器的工作量在减少，而归约操作所带来的通信开销是固定的；而在任务并行模式中只要有足够多的节点和项目集，就能获得较好的负载均衡。

小结 本文针对分布式存储并行计算机体系结构，基于投影树算法，提出了两种序列挖掘模式的并行方法，即数据并行模式和任务并行模式。通过理论分析和实验数据验证可知，相比之下，任务并行模式算法具有更高的性能和较好的可扩展性。

参考文献

1 Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In: Proc. of the Fifth Int'l Conf. on Extending Database Technology, Avignon, France, 1996

2 Agarwall R C, Aggarwal C, Prasad V V V. A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000

3 Agrawal R, Shafer J C. Parallel mining of association rules. IEEE Transactions on Knowledge and Data Eng., 1996, 8(6): 962～969

4 Grama A, Gupta A, Karypis G, Kumar V. Introduction to parallel Computing: Design and Analysis of Algorithms, 2nd Edition. Adison Wesley Publishing Company, Redwood City, CA, 2003

5 Mannila H, Toivonen H, Verkamo A I. Discovering frequent episodes in sequences. In: Proc. of the First Int'l Conf. on Knowledge Discovery and Data mining, Montreal, Quebec, 1995. 210～215

6 Zaki M J. An efficient algorithm for mining frequent sequences. Machine Learning Journal, 2001, 42: 31～60

7 Zaki M J. Scalable algorithms for association mining. Knowledge and Data Engineering, 2000, 12(2): 372～390

8 Pei J, Han J, Mortazavi-Asl B. Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proc. 2001 Int'l Conf. on Data Engineering, 2001

9 Seno M, Karypis G. Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In: IEEE Int'l Conf. on Data Mining, 2002